Attorney Docket No. NWISP030

PATENT APPLICATION

METHODS AND APPARATUS FOR SENDING TARGETED PROBES

Inventor(s): David B. Glasco
10337 Ember Glen Drive
Austin, TX 78726
Citizen of the U.S.

Assignee: Newisys, Inc.
A Delaware corporation

BEYER WEAVER & THOMAS, LLP
P.O. Box 778
Berkeley, California 94704-0778
(510) 843-6200

PATENT
Attorney Docket No. NWISP030

METHODS AND APPARATUS FOR SENDING TARGETED PROBES

CROSS-REFERENCE TO RELATED APPLICATIONS

The present application is related to filed U.S. Application No. 10/288,347 filed on November 4, 2002 and titled Methods And Apparatus For Managing Probe Requests by David B. Glasco, the entirety of which is incorporated by reference herein for all purposes.

BACKGROUND OF THE INVENTION

1. Field of the Invention.

The present invention generally relates to accessing data in a multiple processor system. More specifically, the present invention provides techniques for improving data access efficiency by reducing the number of probes in multiple processor clusters.

2. Description of Related Art

Data access in multiple processor systems raises issues relating to cache coherency. Conventional multiple processor computer systems have processors coupled to a system memory through a shared bus. In order to optimize access to data in the system memory, individual processors are typically designed to work with cache memory. In one example, each processor has a cache that is loaded with data that the processor frequently accesses. The cache is read or written by a processor. However, cache coherency problems arise because multiple copies of the same data can co-exist in systems having multiple processors and multiple cache memories. For example, a frequently accessed data block corresponding to a memory line may be loaded into the cache of two different processors. In one example, if both processors attempt to write new values into the data block at the same time, different data values may result. One value may be written into the first cache while a different value is written into the

second cache. A system might then be unable to determine what value to write through to system memory.

A variety of cache coherency mechanisms have been developed to address such problems in multiprocessor systems. One solution is to simply force all processor writes to go through to memory immediately and bypass the associated cache. The write requests can then be serialized before overwriting a system memory line. However, bypassing the cache significantly decreases efficiency gained by using a cache. Other cache coherency mechanisms have been developed for specific architectures. In a shared bus architecture, each processor checks or snoops on the bus to determine whether it can read or write a shared cache block. In one example, a processor only writes an object when it owns or has exclusive access to the object. Each corresponding cache object is then updated to allow processors access to the most recent version of the object.

Bus arbitration is used when both processors attempt to write the same shared data block in the same clock cycle. Bus arbitration logic decides which processor gets the bus first. Although, cache coherency mechanisms such as bus arbitration are effective, using a shared bus limits the number of processors that can be implemented in a single system with a single memory space.

Other multiprocessor schemes involve individual processor, cache, and memory systems connected to other processors, cache, and memory systems using a network backbone such as Ethernet or Token Ring. Multiprocessor schemes involving separate computer systems each with its own address space can avoid many cache coherency problems because each processor has its own associated memory and cache. When one processor wishes to access data on a remote computing system, communication is explicit. Messages are sent to move data to another processor and messages are received to accept data from another processor using standard network protocols such as TCP/IP. Multiprocessor systems using explicit communication including transactions such as sends and receives are referred to as systems using multiple private memories. By contrast, multiprocessor system using implicit communication including

transactions such as loads and stores are referred to herein as using a single address space.

Multiprocessor schemes using separate computer systems allow more processors to be interconnected while minimizing cache coherency problems. However, it would take substantially more time to access data held by a remote processor using a network infrastructure than it would take to access data held by a processor coupled to a system bus. Furthermore, valuable network bandwidth would be consumed moving data to the proper processors. This can negatively impact both processor and network performance.

Performance limitations have led to the development of a point-to-point architecture for connecting processors in a system with a single memory space. In one example, individual processors can be directly connected to each other through a plurality of point-to-point links to form a cluster of processors. Separate clusters of processors can also be connected. The point-to-point links significantly increase the bandwidth for coprocessing and multiprocessing functions. However, using a point-to-point architecture to connect multiple processors in a multiple cluster system sharing a single memory space presents its own problems.

Consequently, it is desirable to provide techniques for improving data access and cache coherency in systems having multiple clusters of multiple processors connected using point-to-point links.

SUMMARY OF THE INVENTION

According to the present invention, methods and apparatus are provided for increasing the efficiency of data access in a multiple processor, multiple cluster system. Mechanisms for reducing the number of transactions in a multiple cluster system are provided. In one example, owning node information is used to limit the number of probes transmitted in a particular cluster.

In one embodiment, a computer system is provided. The computer system includes a home cluster and a remote cluster. A home cluster includes a first plurality of processing nodes and a home cache coherence controller. The first plurality of processing nodes and the home cache coherence controller are interconnected in a point-to-point architecture. A remote cluster includes a second plurality of processing nodes and a remote cache coherence controller. The remote cache coherence controller is configured to receive a probe from the home cluster, identify a processing node from the second plurality of processing nodes that owns a cache line corresponding to the probe, and send a targeted probe to the processing node.

In another embodiment, a method for providing owning node information is provided. A request for ownership of a memory line is received from a request cluster. The request cluster includes a plurality of request cluster processing nodes. Owning node information associated with the request for ownership is identified at a home cluster. The home cluster comprising a plurality of home cluster processing nodes. Owning node information is maintained in a coherence directory associated with the home cluster.

A further understanding of the nature and advantages of the present invention may be realized by reference to the remaining portions of the specification and the drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention may best be understood by reference to the following description taken in conjunction with the accompanying drawings, which are illustrative of specific embodiments of the present invention.

Figure 1A and 1B are diagrammatic representation depicting a system having multiple clusters.

Figure 2 is a diagrammatic representation of a cluster having a plurality of processors.

Figure 3 is a diagrammatic representation of a cache coherence controller.

Figure 4 is a diagrammatic representation showing a transaction flow for a data access request from a processor in a single cluster.

Figure 5A-5D are diagrammatic representations showing cache coherence controller functionality.

Figure 6 is a diagrammatic representation depicting a transaction flow for a probe request with multiple probe responses.

Figure 7 is a diagrammatic representation showing a cache coherence directory.

Figure 8 is a diagrammatic representation showing a cache coherence directory with owning node information.

Figure 9 is a diagrammatic representation showing an initial ownership request.

Figure 10 is a diagrammatic representation showing a targeted request to an owning node.

Figure 11 is a diagrammatic representation showing a targeted request bypassing a home cluster memory controller.

Figure 12 is a flow process diagram showing the handling of a probe request at a remote cluster cache coherence controller.

DETAILED DESCRIPTION OF SPECIFIC EMBODIMENTS

Reference will now be made in detail to some specific embodiments of the invention including the best modes contemplated by the inventors for carrying out the invention. Examples of these specific embodiments are illustrated in the accompanying drawings. While the invention is described in conjunction with these specific embodiments, it will be understood that it is not intended to limit the invention to the described embodiments. On the contrary, it is intended to cover alternatives, modifications, and equivalents as may be included within the spirit and scope of the invention as defined by the appended claims. Multi-processor architectures having point-to-point communication among their processors are suitable for implementing specific embodiments of the present invention. In the following description, numerous specific details are set forth in order to provide a thorough understanding of the present invention. The present invention may be practiced without some or all of these specific details. Well-known process operations have not been described in detail in order not to unnecessarily obscure the present invention. Furthermore, the present application's reference to a particular singular entity includes that possibility that the methods and apparatus of the present invention can be implemented using more than one entity, unless the context clearly dictates otherwise.

Techniques are provided for increasing data access efficiency in a multiple processor, multiple cluster system. In a point-to-point architecture, a cluster of processors includes multiple processors directly connected to each other through point-to-point links. By using point-to-point links instead of a conventional shared bus or external network, multiple processors are used efficiently in a system sharing the same memory space. Processing and network efficiency are also improved by avoiding many of the bandwidth and latency limitations of conventional bus and external network based multiprocessor architectures. According to various embodiments, however, linearly increasing the number of processors in a point-to-point architecture leads to an exponential increase in the number of links used to connect the multiple processors. In order to reduce the number of links used and to further modularize a multiprocessor system using a point-to-point architecture, multiple clusters are used.

According to various embodiments, the multiple processor clusters are interconnected using a point-to-point architecture. Each cluster of processors includes a cache coherence controller used to handle communications between clusters. In one embodiment, the point-to-point architecture used to connect processors are used to connect clusters as well.

By using a cache coherence controller, multiple cluster systems can be built using processors that may not necessarily support multiple clusters. Such a multiple cluster system can be built by using a cache coherence controller to represent non-local nodes in local transactions so that local nodes do not need to be aware of the existence of nodes outside of the local cluster. More detail on the cache coherence controller will be provided below.

In a single cluster system, cache coherency can be maintained by sending all data access requests through a serialization point. Any mechanism for ordering data access requests is referred to herein as a serialization point. One example of a serialization point is a memory controller. Various processors in the single cluster system send data access requests to the memory controller. In one example, the memory controller is configured to serialize or lock the data access requests so that only one data access request for a given memory line is allowed at any particular time. If another processor attempts to access the same memory line, the data access attempt is blocked until the memory line is unlocked. The memory controller allows cache coherency to be maintained in a multiple processor, single cluster system.

A serialization point can also be used in a multiple processor, multiple cluster system where the processors in the various clusters share a single address space. By using a single address space, internal point-to-point links can be used to significantly improve intercluster communication over traditional external network based multiple cluster systems. Various processors in various clusters send data access requests to a memory controller associated with a particular cluster such as a home cluster. The memory controller can similarly serialize all data requests from the different clusters. However, a serialization point in a multiple processor, multiple cluster system may not be as efficient as a serialization point in a multiple processor, single cluster system.

That is, delay resulting from factors such as latency from transmitting between clusters can adversely affect the response times for various data access requests. It should be noted that delay also results from the use of probes in a multiple processor environment.

Although delay in intercluster transactions in an architecture using a shared memory space is significantly less than the delay in conventional message passing environments using external networks such as Ethernet or Token Ring, even minimal delay is a significant factor. In some applications, there may be millions of data access requests from a processor in a fraction of a second. Any delay can adversely impact processor performance.

According to various embodiments, probe management is used to increase the efficiency of accessing data in a multiple processor, multiple cluster system. A mechanism for eliciting a response from a node to maintain cache coherency in a system is referred to herein as a probe. In one example, a mechanism for snooping a cache is referred to as a probe. A response to a probe can be directed to the source or target of the initiating request. Any mechanism for filtering or reducing the number of probes and probe requests transmitted to various nodes is referred to herein as managing probes. In one example, managing probes entails identifying an owning node associated with the memory line of the probe and sending a probe only to the owning node.

In typical implementations, probe requests are sent to a memory controller that broadcasts probes to various nodes in a system. In such a system, no knowledge of the cache line state is known. All nodes in the system are probed and the request cluster receives a response from each node. In a system with a coherence directory, state information associated with various memory lines can be used to reduce the number of transactions. Any mechanism for maintaining state information associated with various memory lines is referred to herein as a coherence directory. A coherence directory typically includes information for memory lines in a local cluster that are cached in a remote cluster. According to various embodiments, a coherence directory is used to reduce the number of probes to remote quads by inferring the state of local caches. In

other embodiments, a coherence directory is used to eliminate the transmission of a request to a memory controller in a home cluster. A coherence directory can also be used to more accurately send targeted probes. In one example, only a node owning a particular memory line needs to be probed. Information can be added to probe requests and probes to identify the owning node and allow probes to be directed only at owning nodes in a given cluster.

Figure 1A is a diagrammatic representation of one example of a multiple cluster, multiple processor system that can use the techniques of the present invention. Each processing cluster 101, 103, 105, and 107 can include a plurality of processors. The processing clusters 101, 103, 105, and 107 are connected to each other through point-to-point links 111a-f. In one embodiment, the multiple processors in the multiple cluster architecture shown in Figure 1A share the same memory space. In this example, the point-to-point links 111a-f are internal system connections that are used in place of a traditional front-side bus to connect the multiple processors in the multiple clusters 101, 103, 105, and 107. The point-to-point links may support any point-to-point coherence protocol.

Figure 1B is a diagrammatic representation of another example of a multiple cluster, multiple processor system that can use the techniques of the present invention. Each processing cluster 121, 123, 125, and 127 can be coupled to a switch 131 through point-to-point links 141a-d. It should be noted that using a switch and point-to-point links allows implementation with fewer point-to-point links when connecting multiple clusters in the system. A switch 131 can include a processor with a coherence protocol interface. According to various implementations, a multicluster system shown in Figure 1A is expanded using a switch 131 as shown in Figure 1B.

Figure 2 is a diagrammatic representation of a multiple processor cluster, such as the cluster 101 shown in Figure 1A. Cluster 200 includes processors 202a-202d, one or more Basic I/O systems (BIOS) 204, a memory subsystem comprising memory banks 206a-206d, point-to-point communication links 208a-208e, and a service processor 212. The point-to-point communication links are configured to allow interconnections between processors 202a-202d, I/O switch 210, and cache coherence

controller 230. The service processor 212 is configured to allow communications with processors 202a-202d, I/O switch 210, and cache coherence controller 230 via a JTAG interface represented in Fig. 2 by links 214a-214f. It should be noted that other interfaces are supported. It should also be noted that in some implementations, a service processor is not included in multiple processor clusters. I/O switch 210 connects the rest of the system to I/O adapters 216 and 220.

According to specific embodiments, the service processor of the present invention has the intelligence to partition system resources according to a previously specified partitioning schema. The partitioning can be achieved through direct manipulation of routing tables associated with the system processors by the service processor which is made possible by the point-to-point communication infrastructure. The routing tables are used to control and isolate various system resources, the connections between which are defined therein.

The processors 202a-d are also coupled to a cache coherence controller 230 through point-to-point links 232a-d. Any mechanism or apparatus that can be used to provide communication between multiple processor clusters while maintaining cache coherence is referred to herein as a cache coherence controller. The cache coherence controller 230 can be coupled to cache coherence controllers associated with other multiprocessor clusters. It should be noted that there can be more than one cache coherence controller in one cluster. The cache coherence controller 230 communicates with both processors 202a-d as well as remote clusters using a point-to-point protocol.

More generally, it should be understood that the specific architecture shown in Figure 2 is merely exemplary and that embodiments of the present invention are contemplated having different configurations and resource interconnections, and a variety of alternatives for each of the system resources shown. However, for purpose of illustration, specific details of server 200 will be assumed. For example, most of the resources shown in Fig. 2 are assumed to reside on a single electronic assembly. In addition, memory banks 206a-206d may comprise double data rate (DDR) memory which is physically provided as dual in-line memory modules (DIMMs). I/O adapter 216 may be, for example, an ultra direct memory access (UDMA) controller or a small

computer system interface (SCSI) controller which provides access to a permanent storage device. I/O adapter 220 may be an Ethernet card adapted to provide communications with a network such as, for example, a local area network (LAN) or the Internet.

According to a specific embodiment and as shown in Fig. 2, both of I/O adapters 216 and 220 provide symmetric I/O access. That is, each provides access to equivalent sets of I/O. As will be understood, such a configuration would facilitate a partitioning scheme in which multiple partitions have access to the same types of I/O. However, it should also be understood that embodiments are envisioned in which partitions without I/O are created. For example, a partition including one or more processors and associated memory resources, i.e., a memory complex, could be created for the purpose of testing the memory complex.

According to one embodiment, service processor 212 is a Motorola MPC855T microprocessor which includes integrated chipset functions. The cache coherence controller 230 is an Application Specific Integrated Circuit (ASIC) supporting the local point-to-point coherence protocol. The cache coherence controller 230 can also be configured to handle a non-coherent protocol to allow communication with I/O devices. In one embodiment, the cache coherence controller 230 is a specially configured programmable chip such as a programmable logic device or a field programmable gate array.

Figure 3 is a diagrammatic representation of one example of a cache coherence controller 230. According to various embodiments, the cache coherence controller includes a protocol engine 305 configured to handle packets such as probes and requests received from processors in various clusters of a multiprocessor system. The functionality of the protocol engine 305 can be partitioned across several engines to improve performance. In one example, partitioning is done based on packet type (request, probe and response), direction (incoming and outgoing), or transaction flow (request flows, probe flows, etc).

The protocol engine 305 has access to a pending buffer 309 that allows the cache coherence controller to track transactions such as recent requests and probes and associate the transactions with specific processors. Transaction information maintained in the pending buffer 309 can include transaction destination nodes, the addresses of requests for subsequent collision detection and protocol optimizations, response information, tags, and state information.

The cache coherence controller has an interface such as a coherent protocol interface 307 that allows the cache coherence controller to communicate with other processors in the cluster as well as external processor clusters. According to various embodiments, each interface 307 and 311 is implemented either as a full crossbar or as separate receive and transmit units using components such as multiplexers and buffers. The cache coherence controller can also include other interfaces such as a non-coherent protocol interface 311 for communicating with I/O devices. It should be noted, however, that the cache coherence controller 230 does not necessarily need to provide both coherent and non-coherent interfaces. It should also be noted that a cache coherence controller in one cluster can communicate with a cache coherence controller in another cluster.

Figure 4 is a diagrammatic representation showing the transactions for a cache request from a processor in a system having a single cluster without using a cache coherence controller. A processor 401-1 sends an access request such as a read memory line request to a memory controller 403-1. The memory controller 403-1 may be associated with this processor, another processor in the single cluster or may be a separate component such as an ASIC or specially configured Programmable Logic Device (PLD). To preserve cache coherence, only one processor is typically allowed to access a memory line corresponding to a shared address space at anyone given time. To prevent other processors from attempting to access the same memory line, the memory line can be locked by the memory controller 403-1. All other requests to the same memory line are blocked or queued. Access by another processor is typically only allowed when the memory controller 403-1 unlocks the memory line.

The memory controller 403-1 then sends probes to the local cache memories 405, 407, and 409 to determine cache states. The local cache memories 405, 407, and 409 then in turn send probe responses to the same processor 401-2. The memory controller 403-1 also sends an access response such as a read response to the same processor 401-3. The processor 401-3 can then send a done response to the memory controller 403-2 to allow the memory controller 403-2 to unlock the memory line for subsequent requests. It should be noted that CPU 401-1, CPU 401-2, and CPU 401-3 refer to the same processor.

Figures 5A-5D are diagrammatic representations depicting cache coherence controller operation. The use of a cache coherence controller in multiprocessor clusters allows the creation of a multiprocessor, multicluster coherent domain without affecting the functionality of local nodes such as processors and memory controllers in each cluster. In some instances, processors may only support a protocol that allows for a limited number of processors in a single cluster without allowing for multiple clusters. The cache coherence controller can be used to allow multiple clusters by making local processors believe that the non-local nodes are merely a single local node embodied in the cache coherence controller. In one example, the processors in a cluster do not need to be aware of processors in other clusters. Instead, the processors in the cluster communicate with the cache coherence controller as though the cache coherence controller were representing all non-local nodes.

It should be noted that nodes in a remote cluster will be referred to herein as non-local nodes or as remotes nodes. However, non-local nodes refer to nodes not in a request cluster generally and includes nodes in both a remote cluster and nodes in a home cluster. A cluster from which a data access or cache access request originates is referred to herein as a request cluster. A cluster containing a serialization point is referred to herein as a home cluster. Other clusters are referred to as remote clusters. The home cluster and the remote cluster are also referred to herein as non-local clusters.

Figure 5A shows the cache coherence controller acting as an aggregate remote cache. When a processor 501-1 generates a data access request to a local memory

controller 503-1, the cache coherence controller 509 accepts the probe from the local memory controller 503-1 and forwards it to non-local node portion 511. It should be noted that a coherence protocol can contain several types of messages. In one example, a coherence protocol includes four types of messages; data or cache access requests, probes, responses or probe responses, and data packets. Data or cache access requests usually target the home node memory controller. Probes are used to query each cache in the system. The probe packet can carry information that allows the caches to properly transition the cache state for a specified line. Responses are used to carry probe response information and to allow nodes to inform other nodes of the state of a given transaction. Data packets carry request data for both write requests and read responses.

According to various embodiments, the memory address resides at the local memory controller. As noted above, nodes including processors and cache coherence controllers outside of a local cluster are referred to herein as non-local nodes. The cache coherence controller 509 then accumulates the response from the non-local nodes and sends a single response in the same manner that local nodes associated with cache blocks 505 and 507 send a single response to processor 501-2. Local processors may expect a single probe response for every local node probed. The use of a cache coherence controller allows the local processors to operate without concern as to whether non-local nodes exist.

It should also be noted that components such as processor 501-1 and processor 501-2 refer herein to the same component at different points in time during a transaction sequence. For example, processor 501-1 can initiate a data access request and the same processor 501-2 can later receive probe responses resulting from the request.

Figure 5B shows the cache coherence controller acting as a probing agent pair. When the cache coherence controller 521-1 receives a probe from non-local nodes 531, the cache coherence controller 521-1 accepts the probe and forwards the probe to local nodes associated with cache blocks 523, 525, and 527. The cache coherence controller 521-2 then forwards a final response to the non-local node portion 531. In this

example, the cache coherence controller is both the source and the destination of the probes. The local nodes associated with cache blocks 523, 525, and 527 behave as if the cache coherence controller were a local processor with a local memory request.

Figure 5C shows the cache coherence controller acting as a remote memory. When a local processor 541-1 generates an access request that targets remote memory, the cache coherence controller 543-1 forwards the request to the non-local nodes 553. When the remote request specifies local probing, the cache coherence controller 543-1 generates probes to local nodes and the probed nodes provide responses to the processor 541-2. Once the cache coherence controller 543-1 has received data from the non-local node portion 553, it forwards a read response to the processor 541-3. The cache coherence controller also forwards the final response to the remote memory controller associated with non-local nodes 553.

Figure 5D shows the cache coherence controller acting as a remote processor. When the cache coherence controller 561-1 at a first cluster receives a request from a processor in a second cluster, the cache coherence controller acts as a first cluster processor on behalf of the second cluster processor. The cache coherence controller 561-1 accepts the request from portion 575 and forwards it to a memory controller 563-1. The cache coherence controller 561-2 then accumulates all probe responses as well as the data fetched and forwards the final response to the memory controller 563-2 as well as to non-local nodes 575.

By allowing the cache coherence controller to act as an aggregate remote cache, probing agent pair, remote memory, and remote processor, multiple cluster systems can be built using processors that may not necessarily support multiple clusters. The cache coherence controller can be used to represent non-local nodes in local transactions so that local nodes do not need to be aware of the existence of nodes outside of the local cluster.

Figure 6 is a diagrammatic representation depicting the transactions for a data request from a local processor sent to a non-local cluster using a cache coherence controller. The multicluster system includes a request cluster 600, a home cluster 620,

and a remote cluster 640. As noted above, the home cluster 620 and the remote cluster 640 as well as any other clusters excluding the request cluster 600 are referred to herein as non-local clusters. Processors and cache coherence controllers associated with local and non-local clusters are similarly referred to herein as local processors, local cache coherence controllers, non-local processors, and non-local cache coherence controllers, respectively.

According to various embodiments, processor 601-1 in a local cluster 600 sends a data access request such as a read request to a cache coherence controller 603-1. The cache coherence controller 603-1 tracks the transaction in the pending buffer of Figure 3 and forwards the request to a cache coherence controller 621-1 in a home cluster 620. The cache coherence controller 621-1 at the home cluster 620 receives the access request and tracks the request in its pending buffer. In one example, information associated with the requests are stored in the pending buffer. The cache coherence controller 621-1 forwards the access request to a memory controller 623-1 also associated with the home cluster 620. At this point, the memory controller 623-1 locks the memory line associated with the request. In one example, the memory line is a unique address in the memory space shared by the multiple processors in the request cluster 600, home cluster 620, and the remote cluster 640. The memory controller 623-1 generates a probe associated with the data access request and forwards the probe to local nodes associated with cache blocks 625 and 627 as well as to cache coherence controller 621-2.

It should be noted that although messages associated with requests, probes, responses, and data are described as forwarded from one node to another, the messages themselves may contain variations. In one example, alterations are made to the messages to allow the multiple cluster architecture to be transparent to various local nodes. It should be noted that write requests can be handled as well. In write requests, the targeted memory controller gathers responses and sends the responses to the processor when gathering is complete.

The cache coherence controller 641-1 associated with the remote cluster 640 receives a probe from cache coherence controller 621-2 and probes local nodes

associated with cache blocks 645, 647, and 649. Similarly, the cache coherence controller 603-2 associated with the request cluster 600 receives a probe and forwards the probe to local nodes associated with cache blocks 605, 607, and 609 to probe the cache blocks in the request cluster 600. Processor 601-2 receives probe responses from the local nodes associated with cache blocks 605, 607, and 609. It should be noted that a number of cache blocks including 605, 607, 609, 645, 647, and 649 are probed even though not all cache blocks need to be probed. In one instance, only a single cache block holding a memory line in an owned or modified state needs to be probed, because other caches either do not have the memory line cached or have a dirty copy of the memory line. According to various embodiments, the techniques of the present invention provide mechanisms for identifying the owning node and sending targeted probes only to the owning node. A probe directed only at a particular node is referred to herein as a targeted probe. In some examples, targeted probes are sent to owning nodes. Any node holding a copy of a memory line in a owned or modified state is referred to herein as an owning node.

According to various embodiments, cache coherence controller 621-3 accumulates probe responses and sends the probe responses to cache coherence controller 603-3, which in turn forwards the probe responses to the processor 601-3. Cache coherence controller 621-4 also sends a read response to cache coherence controller 603-4, which forwards the read response to processor 601-4. While probes and probe responses carry information for maintaining cache coherency in the system, read responses can carry actual fetched data. After receiving the fetched data, processor 601-4 may send a source done response to cache coherence controller 603-5. According to various embodiments, the transaction is now complete at the requesting cluster 600. Cache coherence controller 603-5 forwards the source done message to cache coherence controller 621-5. Cache coherence controller 621-5 in turn sends a source done message to memory controller 623-2. Upon receiving the source done message, the memory controller 623-2 can unlock the memory line and the transaction at the home cluster 620 is now complete. Another processor can now access the unlocked memory line.

As will be appreciated by one of skill in the art, the specific transaction sequences involving requests, probes, and response messages can vary depending on the specific implementation. In one example, a cache coherence controller 621-3 may wait to receive a read response message from a memory controller 623-1 before transmitting both a probe response message and a read response message to a cache coherence controller 603-3. In other examples, a cache coherence controller may be the actual processor generating the request. Some processors may operate as both a processor and as a cache coherence controller. Furthermore, various data access request messages, probes, and responses associated with reads and writes are contemplated. As noted above, any message for snooping a cache can be referred to as a probe. Similarly, any message for indicating to the memory controller that a memory line should be unlocked can be referred to as a source done message.

It should be noted that the transactions shown in Figure 6 show examples of cache coherence controllers performing many different functions, including functions of remote processors, aggregate local caches, probing agent pairs, and remote memory as described with reference to Figures 5A-5D.

The cache coherence controller 621-1 at the home cluster 620 is acting as a remote processor. When the cache coherence controller receives a request from a request cluster processor, the cache coherence controller is directed to act as the requesting processor on behalf of the request cluster processor. In this case, the cache coherence controller 621-1 accepts a forwarded request from processor 601-1 and sends it to the memory controller 623-1, accumulates responses from all local nodes and the memory controller 623-1, and forwards the accumulated responses and data back to the requesting processor 601-3. The cache coherence controller 621-5 also forwards a source done to the local memory controller 623-2.

The cache coherence controller 603-1 at the request cluster 600 is acting as a remote memory. As remote memory, the cache coherence controller is designed to forward a request from a processor to a proper remote cluster and ensure that local nodes are probed. In this case, the cache coherence controller 603-1 forwards a probe

to cache coherence controller 621-1 at a home cluster 620. Cache coherence controller 603-2 also probes local nodes 605, 607, and 609.

The cache coherence controller 641-1 at the request cluster 640 is acting as a probing agent pair. As noted above, when a cache coherence controller acting as a probing agent pair receives a probe from a remote cluster, the cache coherence controller accepts the probe and forwards it to all local nodes. The cache coherence controller accumulates the responses and sends a final response back to the request cluster. Here, the cache coherence controller 641-1 sends a probe to local nodes associated with cache blocks 645, 647, and 649, gathers probe responses and sends the probe responses to cache coherence controller 621-3 at home cluster 620. Similarly, cache coherence controller 603-2 also acts as a probing agent pair at a request cluster 600. The cache coherence controller 603-2 forwards probe requests to local nodes including local nodes associated with cache blocks 605, 607, and 609.

The cache coherence controller 621-2 and 621-3 is also acting as an aggregate remote cache. The cache coherence controller 621-2 is responsible for accepting the probe from the memory controller 623-1 and forwarding the probe to the other processor clusters 600 and 640. More specifically, the cache coherence controller 621-2 forwards the probe to cache coherence controller 603-2 corresponding to request cluster 600 and to cache coherence controller 641-1 corresponding to remote cluster 640. As noted above, using a multiple cluster architecture may introduce delay as well as other undesirable elements such as increased traffic and processing overhead.

Probes are transmitted to all clusters in the multiple cluster system even though not all clusters need to be probed. For example, if a memory line associated with a probe request is invalid or absent from cache, it may not be necessary to probe all of the caches associated with the various clusters. In a system without a coherence directory, it is typically necessary to snoop all clusters. However, by using a coherence directory, the number of transactions in the system can be reduced by probing only a subset of the clusters in a system in order to minimize traffic and processing overhead.

By using a coherence directory, global memory line state information (with respect to each cluster) can be maintained and accessed by a memory controller or a cache coherence controller in a particular cluster. According to various embodiments, the coherence directory tracks and manages the distribution of probes as well as the receipt of responses. If coherence directory information indicates that probing of a specific cluster is not required, the probe to the specific cluster can be eliminated. In one example, a coherence directory indicates that probing of requesting and remote clusters is not necessary. A cache coherence controller in a home cluster probes local nodes without forwarding probes to the request and remote clusters. The cache coherence controller in the home cluster then sends a response to the request cluster after probe responses are received. However, in typical multiple cluster systems, a requesting cluster expects a predetermined number of responses from the various probed clusters. In one example, if the multiple cluster system includes four clusters, a request cluster would expect probe responses associated with nodes in all four clusters.

According to various embodiments, the techniques of the present invention provide owning node information for the coherence directory. The owning node information allows the transmission of targeted probes. In one example, instead of probing all of the nodes in a particular cluster, the owning node information allows a controller to probe only a single node.

Figure 7 is one example of a typical coherence directory that can be used to allow management and filtering of probes. Various coherence directories are available. In one example, a full directory provides an entry for every memory line in a system. In this example, the coherence directory is maintained at the memory controller and is accessible by a cache coherence controller. However, in a system with a large amount of system memory, a full directory may not be efficient or practical. According to various embodiments, a sparse directory is provided with a limited number of entries associated with a selected set of memory lines. In one example, the coherence directory 701 includes state information 713, dirty data owner information 715, and an occupancy vector 717 associated with the memory lines 711. In some embodiments, the memory line states are modified, owned, shared, and invalid.

In the invalid state, a memory line is not currently available in cache associated with any remote cluster. In the shared state, a memory line may be present in more than one cache, but the memory line has not been modified in any of these caches. When a memory line is in the shared state, an occupancy vector 717 can be checked to determine what caches share the relevant data. An occupancy vector 717 may be implemented as an N-bit string, where each bit represents the availability of the data in the cache of N clusters. Any mechanism for tracking what clusters hold a copy of the relevant memory line in cache is referred to herein as an occupancy vector. The memory line with address 741 is in the shared state, and the occupancy vector 717 indicates that clusters 1 and 3 each have a copy of the shared memory line in cache.

In the modified state, a memory line has been modified and the modified copy exists in cache associated with a particular cluster. When a memory line is modified, dirty data owner information field 715 can be checked to determine the owner of the dirty data. Any mechanism for indicating what cluster owns a modified copy of the memory line in cache is referred to herein as a dirty data owner information field. In one example, the memory line associated with address 781 is modified, and the dirty data owner field 715 indicates that cluster 2 owns the memory line.

In the owned state, a dirty memory line is owned by a single cache but may be resident in multiple caches. It has been read by the owning cache, but has not been modified. In this case, the copy held in memory is stale. If the memory line is in the owned state, dirty data owner field 715 can be accessed to determine which cluster owns the dirty data. In one example, the memory line associated with address 761 is in the owned state and is owned by cluster 4. The occupancy vector 717 can also be checked to determine what other caches may have the relevant data. In this example, the occupancy vector 717 indicates that clusters 2, 3, and 4 each have a copy of the data associated with the memory line in cache.

Although the coherence directory 701 includes the four states of modified, owned, shared, and invalid, it should be noted that particular implementations may use a different set of states. In one example, a system may have the five states of modified,

exclusive, owned, shared, and invalid. The techniques of the present invention can be used with a variety of different possible memory line states.

The coherence directory tracks the various transactions such as probe requests and responses in a multiple cluster system to determine when memory lines are added to the coherence directory, when memory lines are removed from the directory, and when information associated with each memory line is updated. By using the coherence directory, the techniques of the present invention recognize that the number of transactions such as probe requests can be reduced by managing or filtering probes that do not need to be sent to specific clusters.

Figure 8 is one example of a coherence directory including owning node information that can be used to allow management and filtering of probes. In one example, the coherence directory 801 includes state information 813, dirty data owner information 815, owning node information 817, and an occupancy vector 819 associated with the memory lines 811. An owning node is available when a dirty data cluster is identified. The dirty data cluster information 815 allows probing to be limited to a particular cluster of processors. The owning node information 817 allows probing to be limited to a particular node in the cluster.

In the invalid state, a memory line is not currently available in cache associated with any remote cluster. No owning node information 817 is available. In the shared state, a memory line may be present in more than one cache, but the memory line has not been modified in any of these caches. Again, no owning node information 817 is available. In the modified state, a memory line has been modified and the modified copy exists in cache associated with a particular cluster. When a memory line is modified, dirty data owner information field 815 can be checked to determine the owning cluster of the dirty data. Owning node information 817 is provided to identify the particular node having a copy of the memory line in the modified state. Any mechanism for indicating what cluster owns a modified copy of the memory line in cache is referred to herein as a dirty data owner information field. In one example, the memory line associated with address 881 is modified, and the dirty data owner field

815 and the owning node field 817 indicate that cluster 2 has a node 1 that owns the memory line.

In the owned state, a dirty memory line is owned by a single cache but may be resident in multiple caches. It has been read by the owning cache, but has not been modified. In this case, the copy held in memory is stale. If the memory line is in the owned state, dirty data owner field 815 can be accessed to determine which cluster owns the dirty data. Owning node field 817 can be accessed to determine the owning node. In one example, the memory line associated with address 861 is in the owned state and is owned by cluster 4 in node 2. The occupancy vector 817 can also be checked to determine what other caches may have the relevant data. In this example, the occupancy vector 817 indicates that clusters 2, 3, and 4 each have a copy of the data associated with the memory line in cache.

The coherence directory tracks the various transactions such as probe requests and responses in a multiple cluster system to determine when memory lines are added to the coherence directory, when memory lines are removed from the directory, and when information associated with each memory line is updated. By using the coherence directory, the techniques of the present invention recognize that the number of transactions such as probe requests can be reduced by managing or filtering probes that do not need to be sent to specific clusters or nodes.

Figure 9 is a diagrammatic representation showing mechanisms for populating the owning node field in the coherence directory. According to various embodiments, processor 901-1 in a local cluster 900 sends a request to a cache coherence controller 903-1. The cache coherence controller 903-1 tracks the transaction in the pending buffer and forwards the request to a cache coherence controller 921-1 in a home cluster 920. If the request is a request for ownership, such as a read block modify, change to dirty, validate block, etc, the source node information is added to the request before the request is forwarded to the home cluster 920. Any node originating a request for ownership is referred to herein as a request for ownership source node. The cache coherence controller 921-1 at the home cluster 920 receives the access request and tracks the source cluster and the source node in its pending buffer.

Other requests may not include source node information. The cache coherence controller 921-1 forwards the access request to a memory controller 923-1 also associated with the home cluster 920. At this point, the memory controller 923-1 locks the memory line associated with the request. In one example, the memory line is a unique address in the memory space shared by the multiple processors in the request cluster 900, home cluster 920, and the remote cluster 940. The memory controller 923-1 generates a probe associated with the data access request and forwards the probe to local nodes associated with cache blocks 925 and 927 as well as to cache coherence controller 921-2.

When the probe is received at the cache coherence controller 921-2, the source node information is retrieved from the pending buffer and stored in the coherence directory along with the source cluster information. In one example, the coherence directory includes a reference or pointer to the source or owning node. The information can be used to reduce directory storage. In one example, if the directory state is modified, the occupancy vector is no longer needed. In one example, the owning node information is stored in the occupancy vector field.

According to various embodiments, cache coherence controller 921-3 accumulates probe responses and sends the probe responses to cache coherence controller 903-3, which in turn forwards the probe responses to the processor 901-3. Cache coherence controller 921-4 also sends a read response to cache coherence controller 903-4, which forwards the read response to processor 901-4. While probes and probe responses carry information for maintaining cache coherency in the system, read responses can carry actual fetched data. After receiving the fetched data, processor 901-4 may send a source done response to cache coherence controller 903-5. According to various embodiments, the transaction is now complete at the requesting cluster 900. Cache coherence controller 903-5 forwards the source done message to cache coherence controller 921-5. Cache coherence controller 921-5 in turn sends a source done message to memory controller 923-2. Upon receiving the source done message, the memory controller 923-2 can unlock the memory line and the transaction

at the home cluster 920 is now complete. Another processor can now access the unlocked memory line.

Figure 10 is a diagrammatic representation showing mechanisms for using the owning node information to reduce the number of probes in a system. According to various embodiments, processor 1001-1 in a local cluster 1000 sends a data access request such as a read request to a cache coherence controller 1003-1. The cache coherence controller 1003-1 tracks the transaction in the pending buffer and forwards the request to a cache coherence controller 1021-1 in a home cluster 1020. The cache coherence controller 1021-1 at the home cluster 1020 receives the access request and tracks the request in its pending buffer. The cache coherence controller 1021-1 forwards the access request to a memory controller 1023-1 also associated with the home cluster 1020. At this point, the memory controller 1023-1 locks the memory line associated with the request. In one example, the memory line is a unique address in the memory space shared by the multiple processors in the request cluster 1000, home cluster 1020, and the remote cluster 1040. The memory controller 1023-1 generates a probe associated with the data access request and forwards the probe to local nodes associated with cache blocks 1025 and 1027 as well as to cache coherence controller 1021-2.

When the probe is received at the cache coherence controller 1021-2, the coherence directory is accessed to determine if a targeted probe or a directed probe can be sent. In one example, the coherence directory indicates that the memory line is in the owned or modified state. A targeted probe can be used. The owning node and the owning cluster are identified. A targeted probe is then forwarded to the owning cluster 1040 which then forwards the targeted probe to the owning node 1047. The remote cluster cache coherence controller 1041-2 sends a response back to the home cluster 1000. By using a targeted probe, the other nodes in the system are not probed.

Figure 11 is a diagrammatic representation showing mechanisms for using the owning node information to further enhance management of probe requests. According to various embodiments, processor 1101-1 in a local cluster 1100 sends a data access request such as a read request to a cache coherence controller 1103-1. The cache

coherence controller 1103-1 tracks the transaction in the pending buffer and forwards the request to a cache coherence controller 1121-1 in a home cluster 1120. The cache coherence controller 1121-1 at the home cluster 1120 receives the access request and tracks the request in its pending buffer. At this point, the cache coherence controller determines if the memory controller can be bypassed. For example, a memory controller can be bypassed if the transaction is a read on an owned or modified memory line.

There is no need to send a probe request to the requesting cluster or probe local nodes as the owned or modified state implies that the home cluster caches are invalid or shared. A probe is forwarded directly to the owning cluster to acquire the cached data. According to various embodiments, the probe is forwarded with owning node information acquired from the home cluster coherence directory. The remote cluster cache coherence controller 1141-1 receives the targeted probe and identifies the owning node using targeted probe information. The owning node 1147 is probed and a response is received at the cache coherence controller 1141-2. A response is then forwarded back to the home cluster 1100. By using a targeted probe, the other nodes in the multiple cluster system are not needlessly probed.

Figure 12 is a flow process diagram showing probe handling at a remote cluster cache coherence controller. At 1201, a probe associated with a memory line is received from a home cluster. According to various embodiments, the probe may be sent after serialization at a home cluster or the probe may be sent upon bypassing a home cluster memory controller. At 1205, owning node information is identified from the probe. In one example, a probe including owning node information is referred to herein as an augmented probe. At 1209, a probe is sent only to the owning node. Probes do not need to be sent to other nodes in the cluster. It should be noted that in some instances, owning node information may not be provided. If no owning node information is available, the other nodes in the cluster may be probed as well. At 1213, a response is received from the owning node. At 1217, the response is forwarded to the request cluster

While the invention has been particularly shown and described with reference to specific embodiments thereof, it will be understood by those skilled in the art that changes in the form and details of the disclosed embodiments may be made without departing from the spirit or scope of the invention. For example, embodiments of the present invention may be employed with multiple processor clusters connected through a point-to-point, switch, or bus architecture. In another example, multiple clusters of processors may share a single cache coherence controller, or multiple cache coherence controllers can be used in a single cluster. Therefore, the scope of the invention should be determined with reference to the appended claims.